```
Main.Core.eval(Main.Core, :(baremodule Inference
using Core.Intrinsics
import Core: print, println, show, write, unsafe_write, STDOUT,
STDERR
ccall(:jl_set_istopmod, Void, (Bool,), false)

eval(x) = Core.eval(Inference,x)
eval(m,x) = Core.eval(m,x)

include = Core.include

## Load essential files and libraries
include("essentials.jl")
include("generator.jl")
include("reflection.jl")
include("options.jl")

# core operations & types
typealias Cint Int32
typealias Csize_t UInt
include("promotion.jl")
include("tuple.jl")
include("range.jl")
include("expr.jl")
include("error.jl")

# core numeric operations & types
include("bool.jl")
include("number.jl")
include("int.jl")
include("operators.jl")
include("pointer.jl")
const checked_add = +
const checked_sub = -
if !isdefined(Main, :Base)
    # conditional to allow redefining Core.Inference after base exists
    (::Type{T}){T}(arg) = convert(T, arg)::T
end

# Symbol constructors
Symbol(s::String) = Symbol(s.data)
Symbol(a::Array{UInt8,1}) =
    ccall(:jl_symbol_n, Ref{Symbol}, (Ptr{UInt8}, Int32), a, length(a))

# core array operations
include("abstractarray.jl")
include("array.jl")

#TODO: eliminate Dict from inference
include("hashing.jl")
include("nofloat_hashing.jl")

# map-reduce operators
macro simd(forloop)
    esc(forloop)
end
include("reduce.jl")

## core structures
include("intset.jl")
include("dict.jl")
include("iterator.jl")

# core docsystem
include("docs/core.jl")

# compiler
include("inference.jl")
ccall(:jl_set_typeinf_func, Void, (Any,), typeinf_ext)

end # baremodule Inference
))
```

```
baremodule Base
using Core.Intrinsics
ccall(:jl_set_istopmod, Void, (Bool,), true)
include = Core.include
include("coreio.jl")

eval(x) = Core.eval(Base,x)
eval(m,x) = Core.eval(m,x)

include("exports.jl")

## Load essential files and libraries
include("essentials.jl")
include("base.jl")
include("generator.jl")
include("reflection.jl")
include("options.jl")

# core operations & types
include("promotion.jl")
include("tuple.jl")
include("range.jl")
include("expr.jl")
include("error.jl")

# core numeric operations & types
include("bool.jl")
include("number.jl")
include("int.jl")
include("operators.jl")
include("pointer.jl")
include("refpointer.jl")
(::Type{T}){T}(arg) = convert(T, arg)::T
include("checked.jl")
importall .Checked

# vararg Symbol constructor
Symbol(x...) = Symbol(string(x...))

# array structures
include("abstractarray.jl")
include("subarray.jl")
include("array.jl")

# Array convenience converting constructors
(::Type{Array{T}}){T}(m::Integer) = Array{T,1}(Int(m))
(::Type{Array{T}}){T}(m::Integer, n::Integer) = Array{T,2}(Int(m), Int(n))
(::Type{Array{T}}){T}(m::Integer, n::Integer, o::Integer) = Array{T,3}(Int(m),
Int(n), Int(o))
(::Type{Array{T}}){T}(d::Integer...) = Array{T}(convert(Tuple{Vararg{Int}},
d))

(::Type{Vector})() = Array{Any,1}(0)
(::Type{Vector{T}}){T}(m::Integer) = Array{T,1}(Int(m))
(::Type{Vector})(m::Integer) = Array{Any,1}(Int(m))
(::Type{Matrix})() = Array{Any,2}(0, 0)
(::Type{Matrix{T}}){T}(m::Integer, n::Integer) = Matrix{T}(Int(m), Int(n))
(::Type{Matrix})(m::Integer, n::Integer) = Matrix{Any}(Int(m), Int(n))

# TODO: possibly turn these into deprecations
Array{T}(::Type{T}, d::Integer...) = Array{T}(convert(Tuple{Vararg{Int}}, d))
Array{T}(::Type{T}, m::Integer)                       = Array{T,1}(Int(m))
Array{T}(::Type{T}, m::Integer,n::Integer)            = Array{T,2}(Int(m),Int(n))
Array{T}(::Type{T}, m::Integer,n::Integer,o::Integer) = Array{T,3}
(Int(m),Int(n),Int(o))

# numeric operations
include("hashing.jl")
include("rounding.jl")
importall .Rounding
include("float.jl")
include("complex.jl")
include("rational.jl")
include("multinverses.jl")
using .MultiplicativeInverses
include("abstractarraymath.jl")
include("arraymath.jl")

# SIMD loops
include("simdloop.jl")
importall .SimdLoop

# map-reduce operators
include("reduce.jl")

## core structures
include("reshapedarray.jl")
include("bitarray.jl")
include("intset.jl")
include("dict.jl")
include("set.jl")
include("iterator.jl")

# Definition of StridedArray
typealias StridedReshapedArray{T,N,A<:DenseArray}
ReshapedArray{T,N,A}
typealias
StridedArray{T,N,A<:Union{DenseArray,StridedReshapedArray},I<:Tupl
e{Vararg{Union{RangeIndex, AbstractCartesianIndex}}}}
Union{DenseArray{T,N}, SubArray{T,N,A,I},
StridedReshapedArray{T,N}}
typealias
StridedVector{T,A<:Union{DenseArray,StridedReshapedArray},I<:Tuple
{Vararg{Union{RangeIndex, AbstractCartesianIndex}}}}
Union{DenseArray{T,1}, SubArray{T,1,A,I}, StridedReshapedArray{T,1}}
typealias
StridedMatrix{T,A<:Union{DenseArray,StridedReshapedArray},I<:Tuple{
Vararg{Union{RangeIndex, AbstractCartesianIndex}}}}
Union{DenseArray{T,2}, SubArray{T,2,A,I}, StridedReshapedArray{T,2}}
typealias StridedVecOrMat{T} Union{StridedVector{T},
StridedMatrix{T}}

# For OS specific stuff
include(String(vcat(length(Core.ARGS)>=2?Core.ARGS[2].data:"".data,
"build_h.jl".data))) # include($BUILDROOT/base/build_h.jl)
include(String(vcat(length(Core.ARGS)>=2?Core.ARGS[2].data:"".data,
"version_git.jl".data))) # include($BUILDROOT/base/version_git.jl)
include("osutils.jl")
include("c.jl")
include("sysinfo.jl")

# Core I/O
include("io.jl")
include("iostream.jl")
include("iobuffer.jl")

# strings & printing
include("char.jl")
include("intfuncs.jl")
include("strings/strings.jl")
include("unicode/unicode.jl")
include("parse.jl")
include("shell.jl")
include("regex.jl")
include("show.jl")
include("base64.jl")
importall .Base64

# system & environment
include("libc.jl")
using .Libc: getpid, gethostname, time
include("libdl.jl")
using .Libdl: DL_LOAD_PATH
include("env.jl")

# nullable types
include("nullable.jl")

# Scheduling
include("libuv.jl")
include("event.jl")
include("task.jl")
include("lock.jl")

# I/O
include("stream.jl")
include("socket.jl")
include("filesystem.jl")
importall .Filesystem
include("process.jl")
include("multimedia.jl")
importall .Multimedia
include("grisu/grisu.jl")
import .Grisu.print_shortest
include("methodshow.jl")

# core math functions
include("floatfuncs.jl")
include("math.jl")
importall .Math
const (√)=sqrt
const (∛)=cbrt
include("float16.jl")

# multidimensional arrays
include("cartesian.jl")
using .Cartesian
include("multidimensional.jl")
include("permuteddimsarray.jl")
using .PermutedDimsArrays

let SOURCE_PATH = ""
    global include = function(path)
        prev = SOURCE_PATH
        path = joinpath(dirname(prev),path)
        SOURCE_PATH = path
        Core.include(path)
        SOURCE_PATH = prev
    end
end

# reduction along dims
include("reducedim.jl")  # macros in this file relies on string.jl

# basic data structures
include("ordering.jl")
importall .Order
include("collections.jl")

# Combinatorics
include("sort.jl")
importall .Sort

# version
include("version.jl")

function deepcopy_internal end

# BigInts and BigFloats
include("gmp.jl")
importall .GMP
include("mpfr.jl")
importall .MPFR
big(n::Integer) = convert(BigInt,n)
big(x::AbstractFloat) = convert(BigFloat,x)
big(q::Rational) = big(num(q))//big(den(q))

include("combinatorics.jl")

# more hashing definitions
include("hashing2.jl")

# random number generation
include("dSFMT.jl")
include("random.jl")
importall .Random

# (s)printf macros
include("printf.jl")
importall .Printf

# metaprogramming
include("meta.jl")

# enums
include("Enums.jl")
importall .Enums

# concurrency and parallelism
include("serialize.jl")
importall .Serializer
include("channels.jl")
include("clusterserialize.jl")
include("multi.jl")
include("workerpool.jl")
include("pmap.jl")
include("managers.jl")
include("asyncmap.jl")

# code loading
include("loading.jl")

# memory-mapped and shared arrays
include("mmap.jl")
import .Mmap
include("sharedarray.jl")

# utilities - timing, help, edit
include("datafmt.jl")
importall .DataFmt
include("deepcopy.jl")
include("interactiveutil.jl")
include("replutil.jl")
include("test.jl")
include("i18n.jl")
using .I18n

# frontend
include("initdefs.jl")
include("Terminals.jl")
include("LineEdit.jl")
include("REPLCompletions.jl")
include("REPL.jl")
include("client.jl")

# misc useful functions & macros
include("util.jl")

# dense linear algebra
include("linalg/linalg.jl")
importall .LinAlg
const ⋅ = dot
const × = cross
include("broadcast.jl")
importall .Broadcast

# statistics
include("statistics.jl")

# irrational mathematical constants
include("irrationals.jl")

# signal processing
include("dft.jl")
importall .DFT
include("dsp.jl")
importall .DSP

# Numerical integration
include("quadgk.jl")
importall .QuadGK

# Fast math
include("fastmath.jl")
importall .FastMath

# libgit2 support
include("libgit2/libgit2.jl")

# package manager
include("pkg/pkg.jl")
const Git = Pkg.Git

# Stack frames and traces
include("stacktraces.jl")
importall .StackTraces

# profiler
include("profile.jl")
importall .Profile

# dates
include("dates/Dates.jl")
import .Dates: Date, DateTime, now

# sparse matrices, vectors, and sparse linear algebra
include("sparse/sparse.jl")
importall .SparseArrays

# threads
include("threads.jl")
include("threadcall.jl")

# deprecated functions
include("deprecated.jl")

# Some basic documentation
include("docs/helpdb.jl")
include("docs/basedocs.jl")

# Documentation -- should always be included last in sysimg.
include("markdown/Markdown.jl")
include("docs/Docs.jl")
using .Docs, .Markdown
Docs.loaddocs(Core.Inference.CoreDocs.DOCS)

function __init__()
    # Base library init
    reinit_stdio()
    Multimedia.reinit_displays() # since Multimedia.displays uses
STDOUT as fallback
    early_init()
    init_load_path()
    init_parallel()
    init_threadcall()
end

include = include_from_node1
include("precompile.jl")

end # baremodule Base

using Base
importall Base.Operators

Base.isfile("userimg.jl") && Base.include("userimg.jl")
```

coreimg.jl

sysimg.jl

More Standard Library

coreimg.jl

sysimg.jl

But not done yet…

userimg.jl

`--compile-incremental=yes`

```
using MyPackage
 –> Base.require(:MyPackage)
   –> isdefined(Main, :MyPackage) ?
     –> Base._include_from_serialized or Base.include
       –> Base.cachecompile(:MyPackage)
         –> Recursion!
       –> MyPackage.__init__( )
 –> symbols are imported from MyPackage
```